Atty. Docket No. MS306809.1

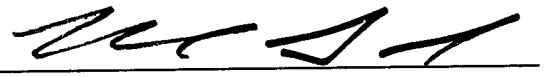
PROMOTION AND DEMOTION TECHNIQUES TO FACILITATE FILE PROPERTY MANAGEMENT BETWEEN OBJECT SYSTEMS

by

Prasanna V. Krishnan, Sambavi Muthukrishnan,
Sameet H. Agarwal, and Balan Sethu Raman

MAIL CERTIFICATION

I hereby certify that the attached patent application (along with any other paper referred to as being attached or enclosed) is being deposited with the United States Postal Service on this date **October 23, 2003**, in an envelope as "Express Mail Post Office to Addressee" Mailing Label Number **EV330022396US** addressed to the Mail Stop Patent Application, Commissioner for Patents, P.O. Box 1450, Alexandria, Virginia 22313-1450.



Himanshu S. Amin

Title: PROMOTION AND DEMOTION TECHNIQUES TO FACILITATE FILE PROPERTY MANAGEMENT BETWEEN OBJECT SYSTEMS

TECHNICAL FIELD

The present invention relates generally to computer systems, and more particularly to a system and method that employs file property handlers to facilitate compatibility between unstructured file property storage in byte streams and structured object representations of the file *via* promotion and demotion of file properties. The term item employed herein generally refers to a structured, schematized object that is stored in a structured object store. A file-backed item to refers to a structured object representation of the file in an object store. The term file can be used to represent an unstructured byte stream that corresponds to a given file-backed item.

BACKGROUND OF THE INVENTION

Traditionally, in a computer file system, a file is the basic unit of data storage. Typically, a file in a file system has the following characteristics. It is a single sequence of bytes. It has a finite length and is stored typically in a non-volatile storage medium. It is created in a directory and has a name that it can be referred to by in file operations, possibly in combination with its path. Additionally, a file system may associate other information with a file, such as permission bits or other file attributes; timestamps for file creation, last revision, and last access *etc.* Specific applications can also store domain-specific properties in the byte stream of the file. For example, files that are used by a word processing application and hence considered as 'documents' may store properties like the Title and Author of the document. These properties are stored within the byte stream of the file in a format that is specific to the application creating the file. The properties are not structured as objects, nor do they have standardized names. The byte streams are unstructured values. Another example would be that a file that stores a music clip has a number of interesting properties such as Genre, Author, Date Recorded, Artist *etc.* stored in the byte stream. In addition to this meta-data, there is a byte stream that

represents the music itself in some universally recognized format. The programming model in dealing with these properties is geared towards manipulating the whole byte stream. The programming model is a bind-reference model which results in a handle being manufactured for the bound instance (the result of a CreateFile/OpenFile call). The subsequent manipulation of the value is done by ReadFile/WriteFile to retrieve and update the relevant portions of the byte stream.

SUMMARY OF THE INVENTION

The following presents a simplified summary of the invention in order to provide a basic understanding of some aspects of the invention. This summary is not an extensive overview of the invention. It is not intended to identify key/critical elements of the invention or to delineate the scope of the invention. Its sole purpose is to present some concepts of the invention in a simplified form as a prelude to the more detailed description that is presented later.

The present invention relates to systems and methods that facilitate file manipulation as a structured object and as a file byte stream by enabling property storage in the byte stream and properties of a structured object representation of a file synchronized when they are independently updated. It is given that there are suitable application programming interfaces for manipulating the file-backed item in an object store and for manipulating the file as an unstructured byte stream. A file property handler is provided to enable unstructured properties in files to be appropriately mapped to and kept consistent with a file-backed item representation for the file *via* system interactions, methods, and procedures referred to as promotion and demotion. Promotion is invoked when an application attempts to modify or manipulate a file by directly manipulating the byte stream corresponding to the file. Thus, promotion is the process whereby the file property handler updates structured file properties in the object store when an application updates unstructured file properties in the byte stream in order to achieve consistency between the two environments. Promotion of the unstructured file properties into an object in the structured object store facilitates various objectives such as, for example:

a) A structured object store allows efficient querying of files based on their properties. This is generally not possible with properties stored in unstructured byte streams.

b) Unstructured properties are represented as well-structured objects that adhere to a standardized object representation. Thus, applications can operate easily with these objects using an object-oriented programming model rather than the model of manipulating the byte stream.

If a promoted file-backed item in the object store is then manipulated and updated, the file property handler facilitates a demotion process, wherein a reverse transformation is performed to update properties in the unstructured file corresponding to the changes in the structured object. In this manner, promotion and demotion enable file properties to be automatically updated and maintained in accordance with the properties suitable for the target system at hand (*e.g.*, update unstructured properties to structured properties *via* promotion and visa versa *via* demotion).

In one aspect of the present invention, a bridge component (*e.g.*, file property manager) communicates with a file property handler to cause transformation of unstructured properties in the file byte stream to a structured object when an application manipulates and saves properties to an unstructured file. The file property manager utilizes mechanisms exposed by the unstructured file store to track modified unstructured files that are to be updated in the object store.

To the accomplishment of the foregoing and related ends, certain illustrative aspects of the invention are described herein in connection with the following description and the annexed drawings. These aspects are indicative of various ways in which the invention may be practiced, all of which are intended to be covered by the present invention. Other advantages and novel features of the invention may become apparent from the following detailed description of the invention when considered in conjunction with the drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a schematic block diagram of a file transformation system in accordance with an aspect of the present invention.

Fig. 2 is a schematic block diagram of more detailed file transformation system in accordance with an aspect of the present invention.

Figs. 3-6 are flow chart diagrams illustrating automated promotion and demotion processes for file property handlers and file property managers in accordance with an aspect of the present invention.

Fig. 7 is a state diagram for an item in accordance with an aspect of the present invention.

Fig. 8 is a schematic block diagram illustrating a suitable operating environment in accordance with an aspect of the present invention.

Fig. 9 is a schematic block diagram of a sample-computing environment with which the present invention can interact.

DETAILED DESCRIPTION OF THE INVENTION

The present invention relates to a system and methodology to facilitate the manipulation of a file as an unstructured byte stream and a structured object by enabling them to be consistent when one is updated independent of the other through specific application programming interfaces.

A file property handler manipulates the unstructured properties in the file in accordance with one or more structured object properties associated with the corresponding file-backed item. A promotion operation is performed to update the file-backed item with structured properties whenever the unstructured file stream is directly updated. Similarly, if the file-backed item were manipulated and updated with application programming interfaces used in the structured object environment, a demotion operation or reverse transformation is performed to update properties in the unstructured representation of the file.

As used in this application, the terms "component," "handler," "manager," "system," and the like are intended to refer to a computer-related entity, either hardware, a combination of hardware and software, software, or software in execution. For example, a component may be, but is not limited to being, a process running on a processor, a processor, an object, an executable, a thread of execution, a program, and/or a computer. By way of illustration, both an application running on a server and the server can be a component. One or more components may reside within a process and/or thread of execution and a component may be localized on one computer and/or distributed between two or more computers.

Referring initially to Fig. 1, a file transformation system 100 is illustrated in accordance with an aspect of the present invention. The system 100 includes a system or application 110 that is typically associated with one or more unstructured files 120, wherein the application 110 and file are generally associated with an unstructured file environment. A bridge module 130 and file property handler 140 are provided to facilitate communications and compatibility with a structured object system or application 150 that is generally associated with a structured object store environment. In one example, the structured system application may be associated with one or more schematized objects at 160 that are typically associated with code such as XML, for example.

If a file were to be modified, saved, and/or manipulated as an unstructured file, the bridge module or file property manager 130 serves to direct the transformation from the unstructured file to the structured object. The transformation is performed by the bridge module invoking the file property handler 140 which would in turn performs a promotion operation to transform unstructured properties of an unmanaged file to structured properties associated with applications that operate against the structured store 160. If the promoted object were to be manipulated from the structured store application, then the file property handler 140 performs a demotion operation which causes properties to be reverse transformed into properties that are then updated in the unstructured file.

In one aspect, promotion operates when a file-based application 110 continues to modify properties by updating a file stream corresponding to a structured object, also known as a file-backed item. Thus, promotion can be modeled as a process of updating a copy of these properties that are in the item in the structured object system 160 in order that the item reflects the changes made by updating the file. In contrast, demotion operates when a new application working against the structured store queries for and modifies items using a structured object Application Programming Interface (API), for example, irrespective of whether the items are file-backed items or not. In the case of file-backed items, some of these properties (which were earlier promoted from the file) are then written back to the file. This process is called demotion, which is in essence a reverse transformation of promotion.

With respect to the file property handler 140, promotion and demotion are achieved by calling a code module, called a File Property Handler (FPH) that determines how to promote properties from and/or demote properties back into a given file. The FPH 140 is typically registered to process one or more selected file extensions. It is noted that the FPH 140 may decide, in some cases, to implement only Promote or Demote functions. As described herein, the term 'promoter' if used refers to an FPH that is being discussed in the context of promotion and 'demoter' as an FPH in the context of demotion.

The bridge component 130 (also referred to as a file property manager) interacts with the FPH 140 to facilitate promotion and demotion. Thus, on receiving an entry from a promotion change queue (described below), the bridge component 130 calls a suitable FPH 140 for a file with a pointer to a stream of the file to be promoted. The FPH 140 then performs extraction and transformation (described below) of the unstructured properties in a file to the structured properties of a managed item that is stored in the structured object store 160. With respect to demotion, a structured object API calls the demoter when an item is updated in the structured object store 160. The demoter employs a reverse transformation and extraction code to write the updated properties back to the file. The extraction portion of the FPH 140 uses an API that is specific to the

file format to extract properties. It is noted that there is generally one registered FPH 140 called for a given file.

Referring now to Fig. 2, a more detailed transformation system 200 is illustrated in accordance with an aspect of the present invention. The system 200 represents a logical model for promotion and/or demotion. In this aspect, one or more FPHs 210 (File Property Handler) reside in managed code. A FPM 220 (File Property Manager) acts as an interoperability or bridge component to the managed FPHs 210. The FPM 220 generally runs as a separate process, distinct from the process maintaining an actual object store 230. As noted above, an abstract BaseFilePropertyHandler class can be provided which defines methods for promotion and demotion. Thus, the File Property Handler 210 is modeled as a concrete managed class deriving from the abstract BaseFilePropertyHandler class that implements methods to be invoked in promotion and/or demotion. For example, the FPM 220 can instantiate the class to invoke promotion for a file with a particular file extension. The following code excerpts represent an example class designation:

```
using System;
using System.Storage;
using System.Storage.Base;

namespace System.Storage.FPM
{
        struct FPMContext
        {
                ItemContext itmCtxt;
                public string  fileExtension;
        };
abstract class BaseFilePropertyHandler
        {
```

```
abstract public void Promote (
    ref Item            itm,          // File item (for write)
    FileStream          fStream,      // File stream (for read)
    PromotionContext    ctx);         // Promotion context

abstract public void Demote (
    Item                existingItm,  // File item (for read)
    FileStream          fStream,      // File stream (for write)
    PromotionContext    ctx);         // Promotion context

abstract public void FirstPromote (
    ref Item            itm,          // File item (for write)
    FileStream          fStream,      // File stream (for write)
    PromotionContext    ctx);         // Promotion context

abstract public void StoreSerializedItem(
    Item                existingItm,  // File item (for read)
    FileStream          fStream,      // File stream (for write)
    PromotionContext    ctx);         // Promotion context

abstract public void RetrieveSerializedItem(
    ref Item            itm,          // File item (for write)
    FileStream          fStream,      // File stream (for read)
    PromotionContext    ctx);         // Promotion context
};
}
```

Promotion is invoked when a file stream of a file-backed item is modified using a File API by applications that work with files. Thus, promotion should be invoked when such an application is used to modify a file in a structured store namespace. There

generally should not be any promotion for files that are not migrated to a structured store namespace and that continue to exist in an unstructured file environment. When a file in the structured store namespace is modified by an application that uses file API, the file promotion manager 220 or service asynchronously invokes the FPH 210 on this file to update the item corresponding to the file. Typically, one FPH 210 is registered per file extension. When there is a pending promotion on a file-backed item, a 'promotionStale' flag on the item is set to 1.

As illustrated in Fig. 2, if an application working with unstructured files 200 modifies a file 240 in the structured store namespace at 230, the FPH 210 performs an extraction and transformation of unstructured properties at 250, before returning the promoted item to be saved at 230. In contrast, demotion is generally invoked when a file-backed item is updated through a structured store API 200. The structured store API 200 allows applications to modify one or more file-backed items or parts of such items. After performing this, when the application tries to save the changed item(s) using the structured store API, the method in the structured store API that is invoked to do the save performs the following: (The method in the structured store API that performs the function of saving the item is conceptually referred to here as the 'save' method, although the exact method name is implementation-dependent.

If a file-backed item, or a part of it is modified through the structured store API 260, the structured store API's save method looks up the demoter corresponding to the file-backed item (based on file extension) and invokes it. The demoter takes in an item (or part of it) for read-write and a file stream for write and updates the file content based on the changes to the item. Demotion is a generally a synchronous operation since it is invoked during the operation that tries to save the item to the structured store. The save method described above updates the file 240 by invoking the demoter as well as write suitable properties to the item in the structured data store.

Figs. 3-6 illustrate automated promotion and demotion processes for file property handlers and file property managers in accordance with an aspect of the present invention. While, for purposes of simplicity of explanation, the methodologies are shown

and described as a series of acts, it is to be understood and appreciated that the present invention is not limited by the order of acts, as some acts may, in accordance with the present invention, occur in different orders and/or concurrently with other acts from that shown and described herein. For example, those skilled in the art will understand and appreciate that a methodology could alternatively be represented as a series of interrelated states or events, such as in a state diagram. Moreover, not all illustrated acts may be required to implement a methodology in accordance with the present invention.

Figs. 3 and 4 are discussed concurrently, wherein Fig. 3 represents processes for a file program manager (FPM) during promotion and Fig. 4 represents processes for a file property handler (FPH) during promotion. In general, Files are queued up for promotion in a queue referred to as a Change Queue. The following are the acts involved in promoting after a file program manager receives a file-backed item from the Change Queue. At 310, the FPM looks-up an FPH for the file by looking up the FPH registration info from a strcutured store that the file is on. At 320, the FPM loads the corresponding FPH. For example, this can be achieved by loading the FPH assembly from a Global Assembly Cache for the respective machine, and retrieving an instance of the FPH class. At 330, the FPM begins a transaction and dequeues the item from the Change Queue. At 340, the FPM retrieves a structured store item (I1) corresponding to the file that was changed. At 350, the FPM performs promotion by calling the appropriate method(s) on the FPH.

After 350, the process proceeds to 410 of Fig. 4 that relates to FPH processing. At this point the FPH may first change the structure of the item. At 420, the FPH extracts properties, and updates the item based on the properties,. At 430, the FPH marks modified parts of the item as promoted. After 430, the process proceeds back to 360 of Fig. 3 for further FPM processing. At 360, the FPM marks the item as not promotionStale. At 370, the FPM applies the changes to the structured object store. At 380, the FPM performs a commit/rollback transaction and closes a file handle.

Figs. 5 and 6 are discussed concurrently, wherein Fig. 5 represents processes for the structured store save function (defined earlier) during demotion and Fig. 6 represents

processes for a file property handler (FPH) during demotion. Proceeding to Fig. 5, the following acts are performed for the item that was modified. At 510, the update function checks if it is a file-backed item. If it is not, demotion code is bypassed and normal processing for a non file-backed item is commenced, otherwise determine the appropriate item for updates performed on only parts of an item which are file-backed. This can be achieved by looking up an isFileBacked bit that is associated with the item in the structured object store.

At 520, the save function queries the store where the file-backed item resides (this could be on another machine in the case of a remote file) to check for the FPH registered for this file extension. This returns details about (e.g. the assembly name & version #) for the FPH that promoted the file. At 530, the update function loads the appropriate FPH (e.g. it may be registered in a Global Assembly Cache of the respective machine) based on the above information. At 540, the save function invokes method(s) on the FPH in order to perform demotion. The changed item also contains a record of the changes that were made to the Item. These changes are tracked by the structured store API.

After 540, the process proceeds to 610 of Fig. 6 for further FPH processing. At 610, the FPH looks at tracked changes in the item to identify what fields need to be demoted. At 620, the FPH reads changes from the item, writes to file properties. This may also include calling the structured store API to read other information from the store. At 630, The FPH marks a 'promoted' flag to 1 on the item or parts of it that are demoted. The process then proceeds back to 550 of Fig. 5 for further save function processing. At 550, the save function applies the changes made to the item to the store. At 560, the save function closes the stream and commits the transaction on the file.

As noted previously, processing may differ with respect to a "first promotion." The act of first promotion is generally distinguished from other promotions. This may be due to that first promotion may need to bring the item and file in sync with each other.

Fig. 7 illustrates a state diagram 700 for a file backed item in accordance with an aspect of the present invention. Given the discussion above, states for a file-backed item are depicted in the diagram 700. A structured store API cannot update a file-backed item

with promotionStale=1. The 'promotionStale' bit on an item is true if it is a file-backed item and there is a promotion pending in the item at 710. An application that desires to update a file-backed item using a structured store API and thus invoke demotion may operate as follows:

1) Fetch the item
2) Query if the item is 'promotionStale'
3) If item is stale
    - call 'SynchronousPromote' method in the API to bring the item up-to-date.

Note: 2 and 3 are optional.

4) Update the fetched item
5) Call the structured store API's save method to try to save the item.
6) If save succeeds in demotion, done.
7) else
8) if error code returned is 'Item Is PromotionStale' (This would be if the app did not perform 2 and 3 above)
    - call 'SynchronousPromote' method in the API to bring the item up-to-date. Returns updatedItem.
        - Go to 4 above to apply the changes on the updatedItem.
9) if error code returned is 'Item Has been Updated'
        - Go to 4 above.

There is in general one FPH registered per file type. However, in the case where a file format is extensible, new properties can be added to the unstructured properties in the file that the FPH was not aware of when it was developed. There are different extensibility schemes supported by the system to allow the FPH for a given file type to be extended to promote/demote the new or custom properties added by software-

vendors/solution providers other than the FPH-writer as well as end-users. This makes it possible to transform the newly added unstructured file properties also into new structured properties on the item. Exemplary extensibility schemes are described below:

1. A software-vendor/solution provider who adds simple unstructured properties to a file and wants these transformed into simple structured properties on the item can do the following:

a) Add corresponding new simple structured properties to the item and

b) Specify declaratively, using an XML scheme, how the newly added unstructured properties should be transformed into the newly added structured item properties.

The main FPH registered for the file type would implement these conversions (during both promotion and demotion) thus not requiring the vendor extending the file to write any new code.

2. A software-vendor/solution provider who adds unstructured properties to a file and wants these transformed into more complex structured properties on the item or establish relationships between items can do the following:

a) Add corresponding complex structured properties to the item and

b) Write code that uses the structured store API to transform the unstructured properties into the newly added structured item properties or establishes relationships between items. This code would be analogous to the implementation of the main FPH for the file type.

3. End-users may add new properties to a file. These are promoted and demoted into a default property set on the structured item representation of the file.

With reference to Fig.8, an exemplary environment 810 for implementing various aspects of the invention includes a computer 812. The computer 812 includes a processing unit 814, a system memory 816, and a system bus 818. The system bus 818 couples system components including, but not limited to, the system memory 816 to the processing unit 814. The processing unit 814 can be any of various available processors.

Dual microprocessors and other multiprocessor architectures also can be employed as the processing unit 814.

The system bus 818 can be any of several types of bus structure(s) including the memory bus or memory controller, a peripheral bus or external bus, and/or a local bus using any variety of available bus architectures including, but not limited to, 16-bit bus, Industrial Standard Architecture (ISA), Micro-Channel Architecture (MSA), Extended ISA (EISA), Intelligent Drive Electronics (IDE), VESA Local Bus (VLB), Peripheral Component Interconnect (PCI), Universal Serial Bus (USB), Advanced Graphics Port (AGP), Personal Computer Memory Card International Association bus (PCMCIA), and Small Computer Systems Interface (SCSI).

The system memory 816 includes volatile memory 820 and nonvolatile memory 822. The basic input/output system (BIOS), containing the basic routines to transfer information between elements within the computer 812, such as during start-up, is stored in nonvolatile memory 822. By way of illustration, and not limitation, nonvolatile memory 822 can include read only memory (ROM), programmable ROM (PROM), electrically programmable ROM (EPROM), electrically erasable ROM (EEPROM), or flash memory. Volatile memory 820 includes random access memory (RAM), which acts as external cache memory. By way of illustration and not limitation, RAM is available in many forms such as synchronous RAM (SRAM), dynamic RAM (DRAM), synchronous DRAM (SDRAM), double data rate SDRAM (DDR SDRAM), enhanced SDRAM (ESDRAM), Synchlink DRAM (SLDRAM), and direct Rambus RAM (DRRAM).

Computer 812 also includes removable/non-removable, volatile/non-volatile computer storage media. Fig. 8 illustrates, for example a disk storage 824. Disk storage 824 includes, but is not limited to, devices like a magnetic disk drive, floppy disk drive, tape drive, Jaz drive, Zip drive, LS-100 drive, flash memory card, or memory stick. In addition, disk storage 824 can include storage media separately or in combination with other storage media including, but not limited to, an optical disk drive such as a compact disk ROM device (CD-ROM), CD recordable drive (CD-R Drive), CD rewritable drive

(CD-RW Drive) or a digital versatile disk ROM drive (DVD-ROM). To facilitate connection of the disk storage devices 824 to the system bus 818, a removable or non-removable interface is typically used such as interface 826.

It is to be appreciated that Fig 8 describes software that acts as an intermediary between users and the basic computer resources described in suitable operating environment 810. Such software includes an operating system 828. Operating system 828, which can be stored on disk storage 824, acts to control and allocate resources of the computer system 812. System applications 830 take advantage of the management of resources by operating system 828 through program modules 832 and program data 834 stored either in system memory 816 or on disk storage 824. It is to be appreciated that the present invention can be implemented with various operating systems or combinations of operating systems.

A user enters commands or information into the computer 812 through input device(s) 836. Input devices 836 include, but are not limited to, a pointing device such as a mouse, trackball, stylus, touch pad, keyboard, microphone, joystick, game pad, satellite dish, scanner, TV tuner card, digital camera, digital video camera, web camera, and the like. These and other input devices connect to the processing unit 814 through the system bus 818 *via* interface port(s) 838. Interface port(s) 838 include, for example, a serial port, a parallel port, a game port, and a universal serial bus (USB). Output device(s) 840 use some of the same type of ports as input device(s) 836. Thus, for example, a USB port may be used to provide input to computer 812, and to output information from computer 812 to an output device 840. Output adapter 842 is provided to illustrate that there are some output devices 840 like monitors, speakers, and printers, among other output devices 840, that require special adapters. The output adapters 842 include, by way of illustration and not limitation, video and sound cards that provide a means of connection between the output device 840 and the system bus 818. It should be noted that other devices and/or systems of devices provide both input and output capabilities such as remote computer(s) 844.

Computer 812 can operate in a networked environment using logical connections to one or more remote computers, such as remote computer(s) 844. The remote computer(s) 844 can be a personal computer, a server, a router, a network PC, a workstation, a microprocessor based appliance, a peer device or other common network node and the like, and typically includes many or all of the elements described relative to computer 812. For purposes of brevity, only a memory storage device 846 is illustrated with remote computer(s) 844. Remote computer(s) 844 is logically connected to computer 812 through a network interface 848 and then physically connected *via* communication connection 850. Network interface 848 encompasses communication networks such as local-area networks (LAN) and wide-area networks (WAN). LAN technologies include Fiber Distributed Data Interface (FDDI), Copper Distributed Data Interface (CDDI), Ethernet/IEEE 1102.3, Token Ring/IEEE 1102.5 and the like. WAN technologies include, but are not limited to, point-to-point links, circuit switching networks like Integrated Services Digital Networks (ISDN) and variations thereon, packet switching networks, and Digital Subscriber Lines (DSL).

Communication connection(s) 850 refers to the hardware/software employed to connect the network interface 848 to the bus 818. While communication connection 850 is shown for illustrative clarity inside computer 812, it can also be external to computer 812. The hardware/software necessary for connection to the network interface 848 includes, for exemplary purposes only, internal and external technologies such as, modems including regular telephone grade modems, cable modems and DSL modems, ISDN adapters, and Ethernet cards.

Fig. 9 is a schematic block diagram of a sample-computing environment 900 with which the present invention can interact. The system 900 includes one or more client(s) 910. The client(s) 910 can be hardware and/or software (*e.g.,* threads, processes, computing devices). The system 900 also includes one or more server(s) 930. The server(s) 930 can also be hardware and/or software (*e.g.,* threads, processes, computing devices). The servers 930 can house threads to perform transformations by employing the present invention, for example. One possible communication between a client 910

and a server 930 may be in the form of a data packet adapted to be transmitted between two or more computer processes. The system 900 includes a communication framework 950 that can be employed to facilitate communications between the client(s) 910 and the server(s) 930. The client(s) 910 are operably connected to one or more client data store(s) 960 that can be employed to store information local to the client(s) 910. Similarly, the server(s) 930 are operably connected to one or more server data store(s) 940 that can be employed to store information local to the servers 930.

What has been described above includes examples of the present invention. It is, of course, not possible to describe every conceivable combination of components or methodologies for purposes of describing the present invention, but one of ordinary skill in the art may recognize that many further combinations and permutations of the present invention are possible. Accordingly, the present invention is intended to embrace all such alterations, modifications and variations that fall within the spirit and scope of the appended claims. Furthermore, to the extent that the term "includes" is used in either the detailed description or the claims, such term is intended to be inclusive in a manner similar to the term "comprising" as "comprising" is interpreted when employed as a transitional word in a claim.